A Secure Java Firewall Tunnel

The JavaServer team at Sun has provided a prototype of some *Secure Firewall Tunneling* technology. This note provides:

- An overview of what the software does,
- A summary of how it works.

What it Does: *The Problem*

Briefly, this firewall tunneling technology supports use of Java technology to support "Extranet" scenarios. It may be viewed as a specialized three-tier system, with the second tier being JavaServer based firewall software. The specific problem being addressed is accessing firewall protected data from the Internet, but more general applications exist.

The link between the first and second tier will normally be over the Internet, and is protected by SSL. This allows sensitive data to be sent over the Internet. The link between the second and third tier will normally be "within" a corporate network, and will not require such cryptographic protections to protect sensitive data.

Sample configurations include NCs (including mobile ones) with Java™ Compatible web browsers supporting the SSL Java Standard Extension API. These will be able to authenticate to firewalls, which grant access to selected applications on the other side of those firewalls. Concurrently, they will be able to access Internet based services without any particular constraints.

User Scenario

In more detail, consider this particular networking scenario:

1. Some *Application User*, is situated on the Internet using some ISP of their own choice using some Java enabled platform.
2. On the boundary between the Internet and a corporate network sits a firewall with special JavaServer software. For concreteness we'll call that machine *firewall.example.com* .
3. Using the web browser, the user contacts a specific *access* URL, which for concreteness we'll call *https://firewall.example.com/access/login*.
4. That URL supports a user *log in*, authenticating with one of a large variety of protocols such as username and password, challenge and response (perhaps with a token card of some kind), and SSL client authentication.
5. Output of that login process is some configuration data which is downloaded into the platform. That includes security configuration, including tunnel routing information, and optional user environment configuration data.
6. Java Firewall Tunnelling software in that platform, potentially including applets and other code that is downloaded as part of login, now knows that to access specific (configured) services, it must go through that firewall. Examples include SMTP and IMAP for e-mail, accessing web based Java applications, and order management.
7. The login session can be terminated by an explicit "logout" action on the part of either the user or the firewall.

For some applications, the ability of this framework to support an *enrollment* process, creating a

new user account if one did not exist previously, is very important. Similarly, the notion that "user accounts" are defined by an extensible "realm" framework is critical to support different kinds of applications.

Significant Features

Some key attributes of this scenario need to be noted:

- This technology supports use of "off the shelf" protocol client software (for example, IMAP/SMTP client applets). The the sole exception is that knowledge of how to punch through firewalls is wrapped up in libraries using Socket Factory APIs as part of the Java Firewall Tunneling (JFT) framework, which are not currently part of the Java core.
- At this time, this only works for TCP based application protocols.
    - Most "mission critical" Internet applications today run over TCP. It has highly desirable scaling properties, due to extensive tuning of TCP stacks. Also, it doesn't require the application developers to master any of the "rocket science" associated with datagram protocols (such as UDP).
    - There are tradeoffs between approaches for supporting application tunneling through the firewall. In some cases, it takes work to make the protocols be "firewall aware". In particular, sites will differ on their willingness to support the "power user" scenario of full IP access to corporate networks:
        - When granting general access through a firewall, there are high risks associated with the threat of client machine subversion. Granting access to file systems, for example, can cause many other security mechanisms to become irrelevant.
        - It is highly desirable to offer control of firewall tunneling through user profiles. For example, most users may have a real need for remote mail access, but only a handful may need access to a particular distributed source code management system.
        - "Virtual Private Network" scenarios tend to call for substantial nonportable low-level modifications to base OS platforms used by Java Virtual Machine implementations.
    - Many protocols, such as SOCKS V5 or custom protocols building directly on SSL, could be used within this framework.
- The user logs in to the firewall.
    - The number of potential users can be very large (entire enterprises of many tens of thousands of users) or small (a handful of users). This a function of the authentication database used to control the firewall login.
    - This isn't by itself a "single signon" scenario, since the different applications may not be using common authentication infrastructure (such as X.509 public key infrastructure).
    - This doesn't necessarily integrate with any authentication which has been done to get access to the Java Compatible platform.

How it Works: *The Solution*

One implementation includes three separate subsystems:

- A web server supporting a "login" infrastructure, configured to use external authentication and user profile services;
- Firewall software builds on that login infrastructure to associate application-specific tunnels with login sessions, within the framework of a particular tunneling protocol.
- Application-specific tunnels are enabled according to data in each user's profile.

The access software supports a particular tunneling protocol, setting up the application specific

tunnels in conjunction with client side *Java Firewall Tunneling* infrastructure.

Client/Firewall Interaction: Login Infrastructure

When the user connects to the access page via HTTPS, that sets up a new SSL connection and an SSL "session". That session is private, and the client knows who the server is. The *login* operation is responsible for getting the server to know who the client is, setting up a login session which grants access (as appropriate) to the various tunnels supported by the firewall, and providing appropriate information to set up the client's environment. The *logout* operation erases all current state for that session on the firewall.

The login user interface can support a variety of schemes, since it is initiated through HTTP. For example, it could use dynamically generated HTML (perhaps using Java servlets), or it could be a custom applet that talks some non-HTTP protocol to the login server.

Preliminary versions of this software support three different authentication schemes, using servlets and dynamically generated HTML:

- **Simple Username/Passphrase authentication**. Two screens are seen by the user in non-error cases: a greeting, holding a form with username and password fields; and the result showing login status.
- **Challenge/Response authentication**. This requires one more screen: a form for entering the username, a form to show the challenge and enter the response, and the result showing login status.
- **SSL client authentication**. This can be done with a single screen, showing login status, since no user interaction should normally be required to use SSL client authentication to contact the firewall.

Additional authentication schemes, perhaps based on JavaCard or other hardware token technology for protecting secrets, could of course be supported.

When the user successfully authenticates, data about that user is retrieved from their profile and associated with their login session. That data includes information about the applications which that user can access through the firewall.

The final step of all successful login processes is the same: a site-customized "access" screen which provides access to various applications, which downloads tunnel and other configuration data into the client, and which supports the option to log out. That tunnel data is contained in a serialized Java bean; it is used to configure the socket factory used when tunneling, and perhaps the specific applications permitted through the firewall. For example, in one tunneling framework it contains a table mapping server host and port names (from the user profile) to the ports on the firewall used to tunnel to that server host and port. (Servers directly on the Internet would be contacted directly, without needing a tunnel.)

At the Client: Application use of Tunneling

Without adopting an "virtual private network" technology, there is no way to eliminate all application-specific work to support tunneling on the client side. That work may be as little as converting to use a socket factory (instead of a constructor) to support transparent connection through a firewall tunnel, but in the case of some protocols could be more.

It is useful to think of some different types of tunneling that need to be done:

1. Simple Client/Server tunneling, where the client may be transparently redirected to go to its server through the firewall;
2. Multi-Server tunneling, where the client needs to talk to more than one server as part of its

setup.
3. Peer-to-Peer tunneling, where the client needs to accept callbacks (on new connections) from servers.

The reason it's important to separate these is that while the first category is extremely simple to set up for tunneling (the client just uses a socket factory API), the second requires some degree of custom support, and the third calls (in general) for some very specialized support (for example, CORBA's application level bridges, perhaps with security policy mediation) which will not be discussed here. For example:

- Access to HTTP services "within" the corporate network will often be problematic because of the use of partially qualified domain names on URLs within that network. This can be addressed by putting HTTP in the second category, rather than the first, and making the URL content handler automatically tunnel to an interior proxy for certain classes of server host name.
- Protocols based on ONC-RPC (such as NFS) need to completely replace the service binding step of the RPC protocol. This is a pure example of the second category.

The tunnel configuration data used by the client must make the client software use these different tunneling approaches.

On the Firewall: Tunneling, Proxies, Full Services

The current form of this work assumes that the firewall should be doing minimal work. Specifically, it just passes data through without much awareness of application protocols: it is a "tunnel".

However, the framework supports configurations where the service to which the client connects does more work than that. There is in fact a spectrum of such configurations. Some interesting points on that spectrum include:

- Running significant network services in the firewall. The HTTP service used to drive "login" here is an example of such a service.
- Running "proxy" services in the firewall. A good example of such work is the kind of application level bridging first described in the CORBA 2.0 interoperability specification, where the firewall supports "policy mediated bridging" to control what ORB services inside the firewall are accessible on the other side.
- The constrained "tunnel" model, just passing data through.

From a security perspective, the less work done on the firewall the better: all that code needs to be treated as the source of a probable attack on the systems inside the firewall. This makes the model of constrained tunnels particularly attractive. However, in some cases the other models have compelling advantages.

Kinds of Application Authentication Protocols

As with tunneling, application protocols sometimes have their own assumptions with respect to authentication, and there are two broad classes: protocols, such as SMTP, which do no authentication; and others, such as IMAP or ONC-RPC based services such as NFS and calendar services, which embed their own. "Real application protocols" tend to be in the latter category, since general purpose client authentication frameworks such as SSL are relatively new.

With respect to IMAP, a variety of options are possible. Three of the most significant options are the use of plaintext passwords, the use of a challenge/response protocol such as OTP or S/Key (RFC 1938), and the use of SSL/TLS.

With respect to ONC RPC services such as NFS, the tunnel data returned by the login process clearly needed to include the UNIX-specific authentication data (UID, GIDs) which is used for that client identification scheme. And that data then needs to be provided to the filesystem client code.

The integration of SSL client authentication with this infrastructure remains to be investigated. There are a variety of configurations, and some of the most natural ones may involve multiple encryptions, which are felt to be undesirable by certain organizations.

Less obvious solutions include "trusted firewalls" which do connection level authentication on behalf of a user; or it might be arranged to tunnel SSL directly to the destination site. Trusted firewalls are often undesirable because they require extending total trust to third parties. These third parties will not as a rule be accountable to the two parties whose security is being protected. However, in some applications the consequent escrowing of encryption and signature keys may be felt to be desirable.

Last modified: *February 23, 1998*
Author: *David.Brownell@eng.sun.com*